Fork-Join and Data-Driven Execution Models on Multi-Core Architectures: Case study of the FMM

Abdelhalim Amer[1] Naoya Maruyama[2] Miquel Pericàs[1]
Kenjiro Taura[3] Rio Yokota[4] Satoshi Matsuoka[1]

[1]Tokyo Institute of Technology, Tokyo, Japan

[2]RIKEN, Kobe, Japan

[3]The University of Tokyo, Tokyo, Japan

[4]KAUST, Saudi Arabia

ISC'13, Leipzig, Germany

Outline

- Introduction
  - The Fork-Join Model
  - The Data-Driven Model
  - Trade-Off: Data locality vs. idle times
  - The Fast Multipole Method (FMM)
- FMM Implementations
  - Fork-Join FMM
  - Data-Driven FMM
- Performance Evaluation and Analysis
  - Test-bed Configuration
  - The Fork-Join FMM bottlenecks at scale
  - Comparative Analysis
  - Memory-intensive Kernel Analysis
- Conclusion and Future Work

Outline

- Introduction
  - The Fork-Join Model
  - The Data-Driven Model
  - Trade-Off: Data locality vs. idle times
  - The Fast Multipole Method (FMM)
- FMM Implementations
  - Fork-Join FMM
  - Data-Driven FMM
- Performance Evaluation and Analysis
  - Test-bed Configuration
  - The Fork-Join FMM bottlenecks at scale
  - Comparative Analysis
  - Memory-intensive Kernel Analysis
- Conclusion and Future Work

Introduction

Programming parallel machines is complex

- Extract parallelism; while
- Minimizing data movements

Execution models:

- **Fork-Join (Bulk-Synchronous)**: promotes data locality and tolerates idle times
- **Data-Driven (Asynchronous)**: keeps processors busy to the detriment of data locality

$\Rightarrow$Trade-off: **data locality** vs. **minimizing idle times**

Proposition

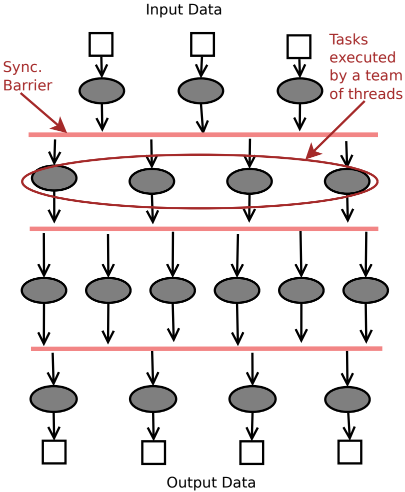
Study this trade-off on Multi-Cores + the Fast Multipole Method (FMM)

Outline

- Introduction
  - The Fork-Join Model
  - The Data-Driven Model
  - Trade-Off: Data locality vs. idle times
  - The Fast Multipole Method (FMM)
- FMM Implementations
  - Fork-Join FMM
  - Data-Driven FMM
- Performance Evaluation and Analysis
  - Test-bed Configuration
  - The Fork-Join FMM bottlenecks at scale
  - Comparative Analysis
  - Memory-intensive Kernel Analysis
- Conclusion and Future Work

The Fork-Join Model

- Execution = multiple steps synchronized by global barriers
- Each step is executed in parallel
- A step may work on a subset of data $\rightarrow$ Possibility to exploit data locality
- We do not consider nested fork-join

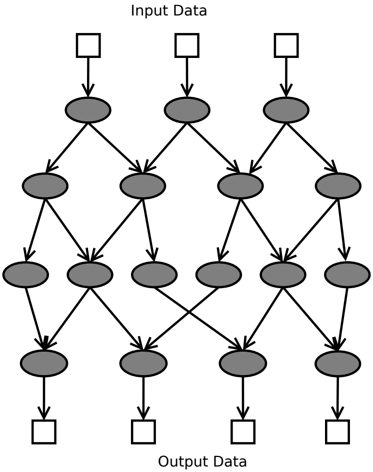


Outline

- Introduction
  - The Fork-Join Model
  - The Data-Driven Model
  - Trade-Off: Data locality vs. idle times
  - The Fast Multipole Method (FMM)
- FMM Implementations
  - Fork-Join FMM
  - Data-Driven FMM
- Performance Evaluation and Analysis
  - Test-bed Configuration
  - The Fork-Join FMM bottlenecks at scale
  - Comparative Analysis
  - Memory-intensive Kernel Analysis
- Conclusion and Future Work

The Data-Driven Model

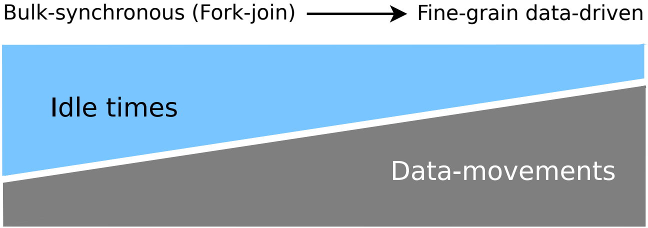
- Breaks global synchronizations into fine-grain local synchronizations
- Runtimes and schedulers extract parallelism and minimize idle times
- Difficult to express locality and possible loss in cache performance



Outline

Data locality vs. Idle times trade-off

Parallel execution models exhibit a trade-off between data-locality and computational units idle times:

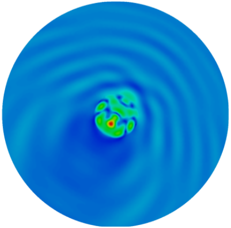


$\Rightarrow$ We study the extreme cases: **Bulk-Synchronous** vs. **Fine-grain data-driven** methods

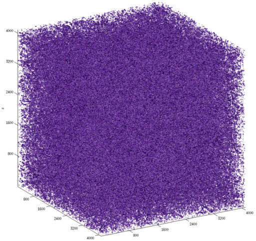
Outline

The Fast Multipole Method

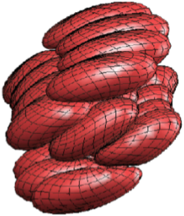
- Solves n-body problems with $O(N)$ complexity
- Used in many scientific simulations:



Elctrodynamics [1]



Fluid dynamics [2]



Blood flow [3]

---

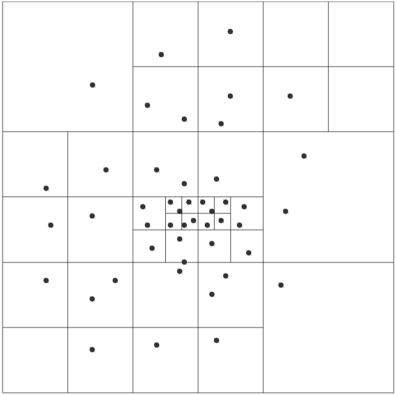
[1] S. Chaillat, M. Bonnet, J.F. Semblat: A multi-level fast multipole bem for 3-d elastodynamics in the frequency domain. Computer Methods in Applied Mechanics and Engineering 197 (2008)

[2] R. Yokota, T. Narumi, L.A. Barba, K. Yasuoka: Petascale turbulence simulation using a highly parallel fast multipole method. (2011)

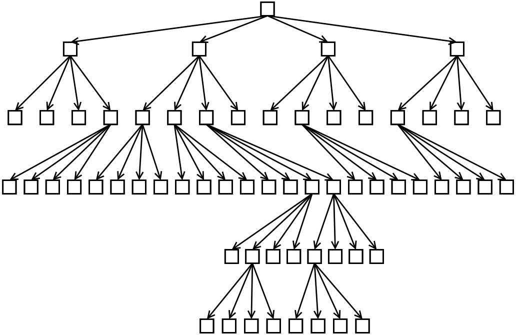
[3] A. Rahimian, I. Lashuk, S. Veerapaneni, A. Chandramowlishwaran, D. Malhotra, L. Moon, R. Sampath, A. Shringarpure, J. Vetter, R. Vuduc, D. Zorin, and G. Biros, Petascale Direct Numerical Simulation of Blood Flow on 200K Cores and Heterogeneous Architectures, SC 2010

Basics of FMMs: Domain decomposition

2D domain decomposition example

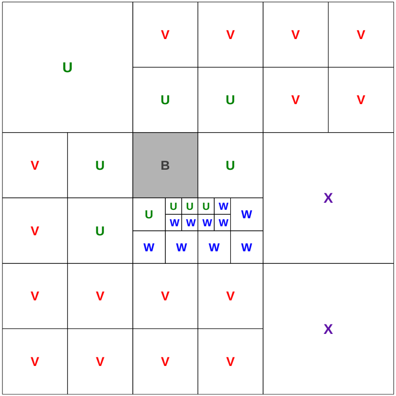


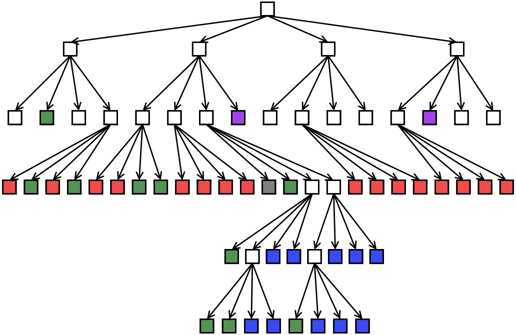
Corresponding quad-tree



Basics of FMMs: Interaction lists

Interaction lists for a target box B in a quad-tree[1]

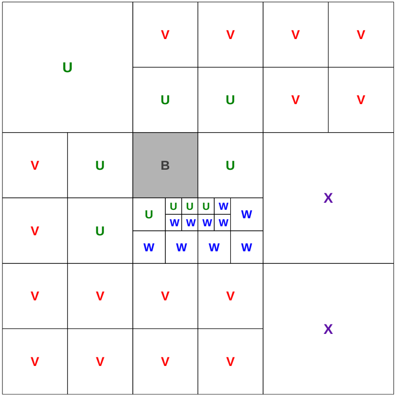




[1] A. Chandramowlishwaran, S. Williams, L. Oliker, I. Lashuk, G. Biros, R. Vuduc: Optimizing and tuning the fast multipole method for state-of-the-art multicore architectures". IPDPS (2010)

Basics of FMMs: Interaction lists

Interaction lists for a target box B in a quad-tree[1]



1. **Near field direct evaluation**
   - **U-list**: Compute intensive
2. **Far field approximation**
   - **Upward**: Parent-children dependencies
   - **V-list**: Memory intensive
   - **X and W-lists**: High workload variation
   - **Downward**: Parent-children dependencies

[1] A. Chandramowlishwaran, S. Williams, L. Oliker, I. Lashuk, G. Biros, R. Vuduc: Optimizing and tuning the fast multipole method for state-of-the-art multicore architectures". IPDPS (2010)

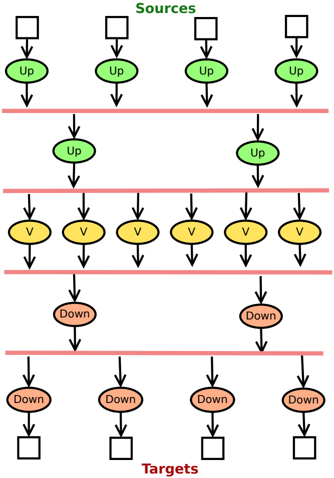
Outline

- Introduction
  - The Fork-Join Model
  - The Data-Driven Model
  - Trade-Off: Data locality vs. idle times
  - The Fast Multipole Method (FMM)
- FMM Implementations
  - Fork-Join FMM
  - Data-Driven FMM
- Performance Evaluation and Analysis
  - Test-bed Configuration
  - The Fork-Join FMM bottlenecks at scale
  - Comparative Analysis
  - Memory-intensive Kernel Analysis
- Conclusion and Future Work

Outline

Fork-Join implementation of the FMM[1]

- Each step implemented with OpenMP work-sharing constructs
- Upward and Downward: level-by-level synchronization barriers
- U-list and V-list: manual partitioning for improved load-balancing
- X and W-list: OpenMP `static` scheduler



[1] A. Chandramowlishwaran, S. Williams, L. Oliker, I. Lashuk, G. Biros, R. Vuduc: Optimizing and tuning the fast multipole method for state-of-the-art multicore architectures. IPDPS (2010)

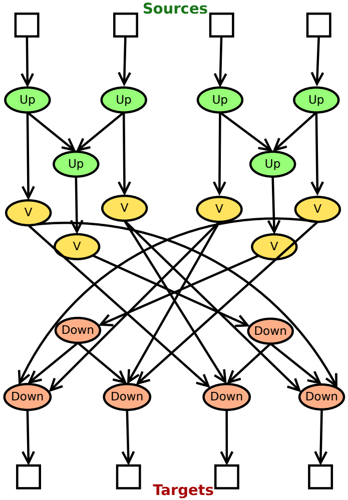
Outline

- Introduction
  - The Fork-Join Model
  - The Data-Driven Model
  - Trade-Off: Data locality vs. idle times
  - The Fast Multipole Method (FMM)
- FMM Implementations
  - Fork-Join FMM
  - Data-Driven FMM
- Performance Evaluation and Analysis
  - Test-bed Configuration
  - The Fork-Join FMM bottlenecks at scale
  - Comparative Analysis
  - Memory-intensive Kernel Analysis
- Conclusion and Future Work

Data-Driven implementation of the FMM

Data-Driven FMMs related work:

- Based on task schedulers: Quark[1], StarPU[2], and others
- Overhead: task management + data dependency tracking



Proposition

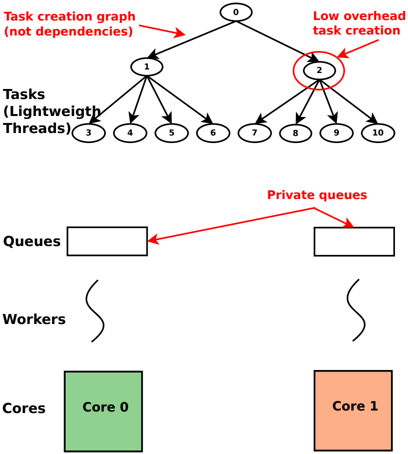
- Lightweight threads: low overhead task management
- Manual synchronization: atomic counters + task nesting

[1] Ltaief, H., Yokota, R.: Data-driven execution of fast multipole methods. (2012)

[2] Agullo, E., Bramas, B., Coulaud, O., Darve, E., Messner, M., Takahashi, T.:Pipelining the fast multipole method over a runtime system. (2012)

MassiveThreads library[1]

- Cilk[2]-like runtime: **Work-first** scheduling with inter-worker **work-stealing**
- Low overhead task management
- Private queues per worker which enables **Distributed Scheduling**



[1] http://code.google.com/p/massivethreads/
[2] http://supertech.csail.mit.edu/cilk/

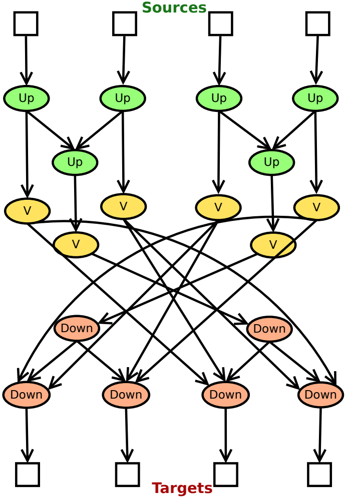
Data-Driven FMM: implementation details

Fine-grain tasks where each task:

- Operates at the tree node level
- Is embedded in a lightweight thread
- May recursively create other tasks which enables **subtree working-sets**

A Task has two parts:

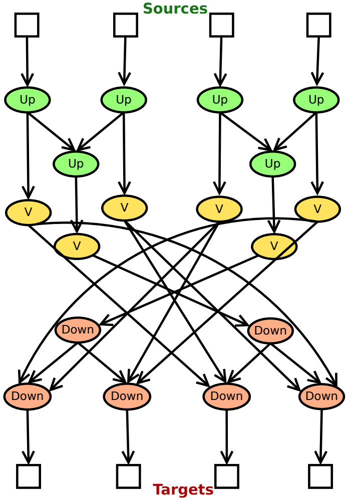
- Computation
- Synchronization in two steps
  - Update sync. counters
  - Dependent task creation



Data-Driven FMM: implementation details

```
void* V (src){
  for(trg in Vlist(src))
  {

     compute_V(trg,src);

     trg.down_counter++;

     if(trg.down_counter = nb_dep(trg))
       create_task(Down, trg);

  }
}
```

- Computation
- Synchronization in two steps
  - Update sync. counters
  - Dependent task creation



Outline

- Introduction
  - The Fork-Join Model
  - The Data-Driven Model
  - Trade-Off: Data locality vs. idle times
  - The Fast Multipole Method (FMM)
- FMM Implementations
  - Fork-Join FMM
  - Data-Driven FMM
- Performance Evaluation and Analysis
  - Test-bed Configuration
  - The Fork-Join FMM bottlenecks at scale
  - Comparative Analysis
  - Memory-intensive Kernel Analysis
- Conclusion and Future Work

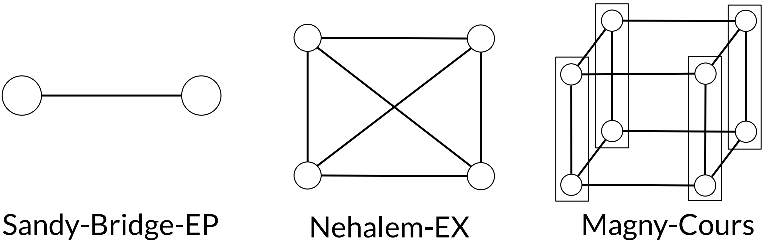
Outline

- Introduction
  - The Fork-Join Model
  - The Data-Driven Model
  - Trade-Off: Data locality vs. idle times
  - The Fast Multipole Method (FMM)
- FMM Implementations
  - Fork-Join FMM
  - Data-Driven FMM
- Performance Evaluation and Analysis
  - Test-bed Configuration
  - The Fork-Join FMM bottlenecks at scale
  - Comparative Analysis
  - Memory-intensive Kernel Analysis
- Conclusion and Future Work

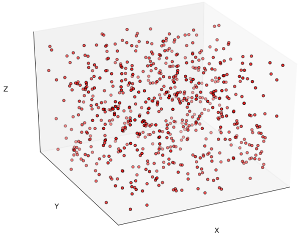
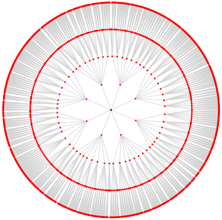
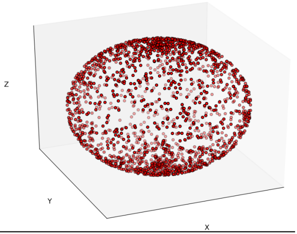
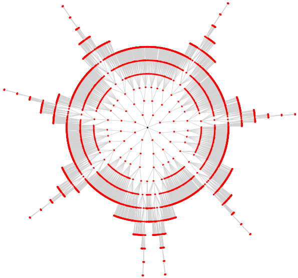
Target Multi-Core Architectures

	Sandy-Bridge-EP	Nehalem-EX	Magny-Cours
Processor	Xeon E5-2620	Xeon X7550	Opteron 6172
CPU Frequency (Ghz)	2.0	2.0	2.1
#NUMA-nodes×#Cores	2×6	4×8	8×6
L3 Cache size (MB)	15	18	6
Total Memory BW (MB/s)	52590.4	68827.3	74720.4

NUMA nodes topology for each machine



Simulation Input[1]

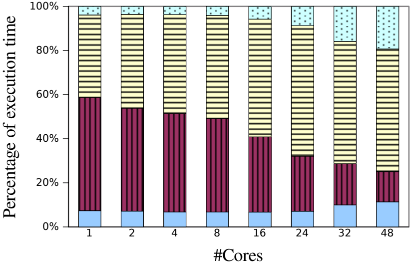
	Particle Distribution	Corresponding Oct-tree
Uniform		
Elliptical		

[1] The particle distribution and oct-tree figures were obtained with a small problem size for simplicity reasons. For the other experiments, 4 millions particles were used with 250 particles per boxe.

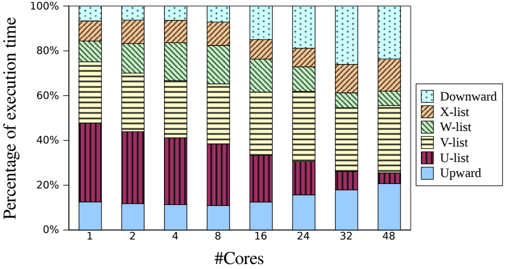
Outline

- Introduction
  - The Fork-Join Model
  - The Data-Driven Model
  - Trade-Off: Data locality vs. idle times
  - The Fast Multipole Method (FMM)
- FMM Implementations
  - Fork-Join FMM
  - Data-Driven FMM
- Performance Evaluation and Analysis
  - Test-bed Configuration
  - The Fork-Join FMM bottlenecks at scale
  - Comparative Analysis
  - Memory-intensive Kernel Analysis
- Conclusion and Future Work

The Fork-Join FMM bottlenecks at scale



(a) Uniform distribution



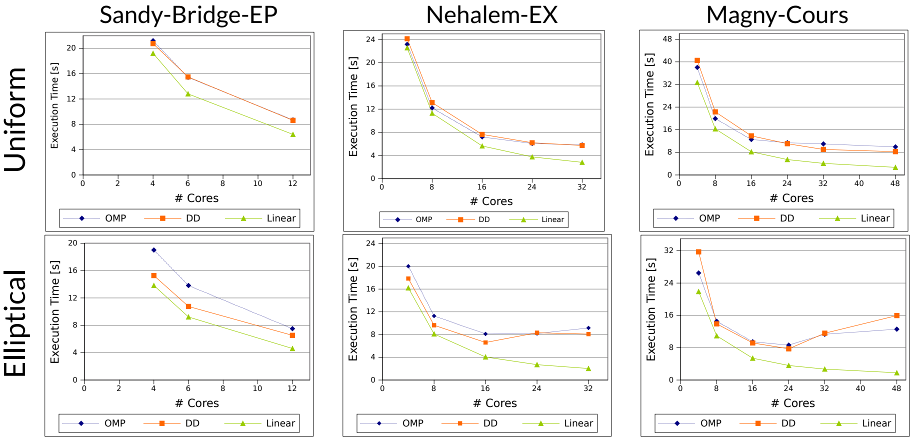
(b) Elliptical distribution

- Single thread execution: U-list and V-list are bottlenecks
- Larger scale: in addition to V-list Upward and Downward (often neglected) consume more time than U-list
- Need an optimized implementation for each stage

Outline

- Introduction
  - The Fork-Join Model
  - The Data-Driven Model
  - Trade-Off: Data locality vs. idle times
  - The Fast Multipole Method (FMM)
- FMM Implementations
  - Fork-Join FMM
  - Data-Driven FMM
- Performance Evaluation and Analysis
  - Test-bed Configuration
  - The Fork-Join FMM bottlenecks at scale
  - Comparative Analysis
  - Memory-intensive Kernel Analysis
- Conclusion and Future Work

Comparative strong scaling



The data-driven method, as compared to the original design:

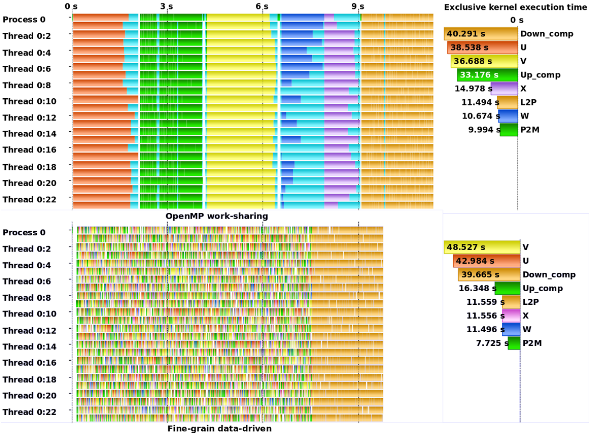
- Gives similar performance for a uniform distr.
- Scales better for the irregular distr. (except in the case of Magny-Cours at high core count, likely due to the small cache size)

Under the hood: Execution trace

Data-driven method results:

- Global synchronization eliminated
- Upward kernels faster (better data reuse)
- V-list kernels slower (likely cache contention)

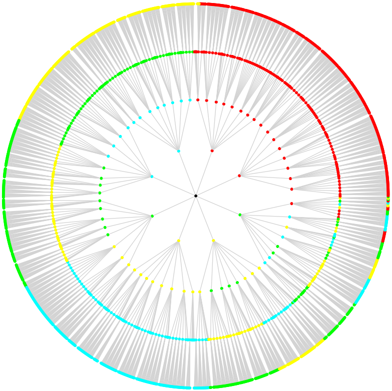
Execution trace on the Magny-cours machine



⇒ The data-driven execution does not address the memory intensive kernel bottleneck, but makes it worse!

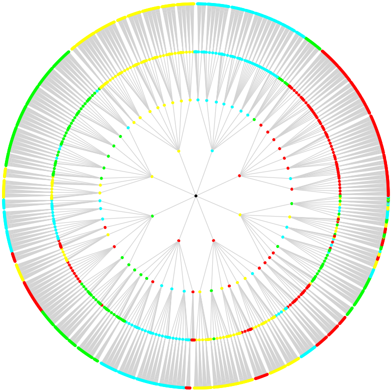
Why Upward has a better data locality?

Uniform Oct-trees, color = thread



OpenMP with a guided scheduler

⇒Potential lose of inter-level data locality

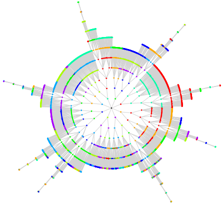


Data-Driven

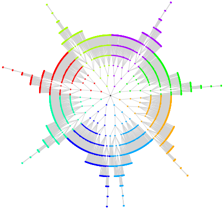
⇒High inter-level data locality

Why Upward has a better data locality?

Irregular Oct-tree, color = thread



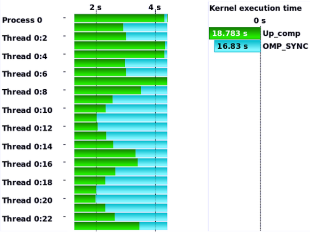
Guided scheduler with 8 threads



Sub-tree partitioning with 8 threads

Trace with an Elliptical distr.

OpenMP for guided scheduler



Manual sub-tree partitioning

$\Rightarrow$ Better to keep data local and have more idle times than being dynamic and increase data movements!

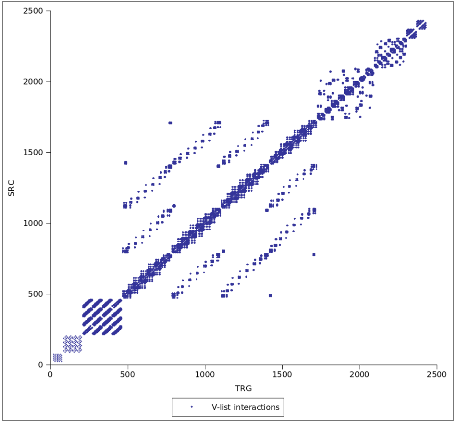
Outline

- Introduction
  - The Fork-Join Model
  - The Data-Driven Model
  - Trade-Off: Data locality vs. idle times
  - The Fast Multipole Method (FMM)
- FMM Implementations
  - Fork-Join FMM
  - Data-Driven FMM
- Performance Evaluation and Analysis
  - Test-bed Configuration
  - The Fork-Join FMM bottlenecks at scale
  - Comparative Analysis
  - Memory-intensive Kernel Analysis
- Conclusion and Future Work

V-list source-target interactions

- Reads from a source vector and writes into a target vector
- Source-target vector elements relationship: sparse matrix
- Sparse data access pattern in non-NUMA aware fashion

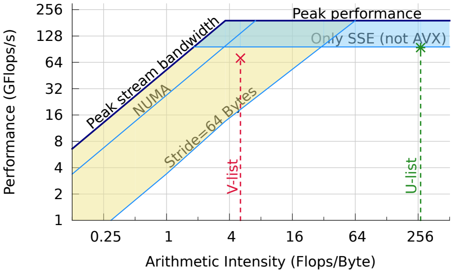
V-list interactions in an Elliptical distr.



Roofline Model Analysis

- Arithmetic intensity and GFlops: performance counters
- Bandwidth roof and ceilings: Stream benchmark[1]
- V-list performance limited by the bandwidth ceilings
- Currently the main bottleneck for both parallel execution methods

Roofline plot for the Sandy-Bridge-EP Machine



[1] McCalpin, J.D.: Memory bandwidth and machine balance in current high performance computers. IEEE Computer Society Technical Committee on Computer Architecture TCCA Newsletter (1995)

Outline

Conclusion and Future Work

- Low overhead fine-grain Data-Driven execution of FMM using distributed task scheduling
- Data-Driven showed a better trade-off between data locality and synchronization overheads
- This method made worse the memory intensive kernel execution

Future work:

- More tuning can be performed
  - Tuning the task granularity
  - Hiding V-list memory latency by the other computations
  - Blocking source data in V-list
- Enlarge the study to other irregular algorithms and many-core architectures
- Building runtimes which take into account the costs of data-movements and idle times